**Introduction:** Types of languages – Procedural, Functional, Object Oriented

Object-> code + data

**Static vs Dynamic Languages:**

* **Static:** Type check at compile time
  + Errors will show at compile time
  + More control
  + Declare datatype before you use it
* **Dynamic:** Perform type check at runtime
  + Error might not show till program is run
  + No need to declare data type of variables
  + Might give error at compile time

**Introduction to JAVA:**

.java file -> **compile**-> .class file (byte code) -> **Interpreter** (line by line)->M/C code (0,1)

**Java is a Strongly typed language:** It checks data type at compile time.

**Byte Code:**

We need JVM to run this code and converted to machine code.

Can run on all operating system.

**Architecture of Java:**

JDK = JRE + Development tools

JRE = JVM + Library Classes

JVM has JIT

**Static:**

Run function without creating object of the class.

**main:**

Entry point of java code

**String[] args:**

Command line arguments, we can pass these arguments from cmd

java Main “Devarshi”

**Change Bytecode Location:**

javac -d . Demo.java

javac -d .. Demo.java

**Package:**

Folder to locate java files

And to provide access

**Primitive Data Types in Java:**

Any data type which cannot be break into any other data type.

Int a = 10;-> 10 is literal, a is an identifier

**Type Casting:**

Float > integer

If asking for integer but giving float will not work but vice versa will work.

Int num = (int)(67.56f); 🡪 type casting 🡪 67

Ex: int a = 257;

byte b = (byte)(a);

b 🡪 1 (max value of byte is 256 so 1 is remainder of maximum value)

**Java follows UNICODE principle**

**Garbage Collection:**

Java does memory management automatically behind the scene.

If any object is not referencing any variable, it becomes part of garbage collection.

Create variable in smallest scope so if we don’t need that variable garbage collector will remove it quickly.

**Working:**

Young generation heap holds all objects that were created very recently.

And as heap start getting full it does mark and sweep, it finds object that do not have reference variable and sweep them.

Object that has reference variable for long moves from young generation heap to old generation heap. Java does mark and sweep very less often for old generation heap memory.

**Static Vs Non-Static Variables and Methods:**

Static method does not require individual object.

Non static method needs to create individual object of Class and access it for all non-static variables and methods.

Inside the static method we are not allowed to access any non-static variable.

In static we are dealing on class level not to individual object level.

Technically we can access static method and variable in non-static method.

**Type promotion rules:**

Two types operation will give bigger data type return value

EX:

byte b = 42;

char c = ‘a’;

short s = 1024;

int I = 50000;

float f = 5.67f

double d = 0.1234;

double result = (f\*b) + (i/c) – (d-s);

* Float value 1777.016

**While loop:**

**For loop:**

**[Q1\_LargestNumber.java](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q1_LargestNumber.java)**

**[Q2\_AlphabetCaseCheck](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q2_AlphabetCaseCheck.java)**

**[Q3\_FibonacciNumbers](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q3_FibonacciNumbers.java)**

[**Q4\_CountingOccurrences**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q4_CountingOccurrences.java)

[**Q5\_ReverseOfNumber**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q5_ReverseOfNumber.java)

Q6\_Calculator

**Switch Statements:**

switch(val){

case val1:

//do something

Break;

Case val2:

// do something

breake;

default:

//do something

}

* If break is not provided code will execute till the end

**Functions/Methods in Java:**

**Scope:**

Accessing variable locally or globally.

Block Scope: variable within a scope

Shadowing: changing global variable value in a block / it will hide global value of that variable

**Variable Arguments(Varargs):**

when to pass n number of arguments.

static void func(int …v){

// …v -> it will internally store in an array

System.out.println(Arrays.toString(v)); 🡪 [1,2,3 . . . etc]

}

static void func(String a, int …v){

// first pass string then pass n number of integers

// order is important

}

**Overloading:**

Two or more functions exists with same name and with different parameters.

[**Q7\_PrimeNumber**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q7_PrimeNumber.java)

[**Q8\_ArmStrongNumbers**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q8_ArmStrongNumbers.java)

**Arrays and Array List:**

Syntax: datatype [] variablName = new datatype[size];

datatype [] variableName = {val1, val2, val3};

collection of same data type.

int [] i = new int [5];

int [] i 🡪 reference variable/ declaration of array (i is getting defined in the stack) (compile time)

new int [5] 🡪 initialisation/ actually here object is being created in heap (run time/ dynamic memory allocation)

# Primitives are stored in stack memory

# Heap objects are not continuous (allocation of array data location in heap is not continuous)

# In java internally array objects may not be continuous -> depends on JVM

# position of array starts from zero

**String Array:**

String [] arr = new String [4];

arr [0] 🡪 null

arr-> store in stack memory

[\_, \_, \_, \_] -> 4 elements stored in heap and this each element itself is an object which has stored in different part of memory

All reference variables by default points to “null”

**Length:**

arr. length

**ForEach Loop:**

for (int num: arr) {

// for every element in array, num represents element of array

}

**Print Array:** Arrays.toString(arr);

**2D Arrays:**

Int [] [] arr2D = new int [3] [];

Length of column is not mandatory

Int [] [] arr2D = {

{},{},{}

}

**Dynamic Arrays:**

**Array List:** if we don’t know the size of array

(it is similar to vectors in C++)

ArrayList<String> list = new ArrayList<>(initialcapacity:10);

In peranthesis we can give initial length

<> -> in this we cannot pass primitive , can pass wrapper class only

list.add(“Hello”);

list.add(“world”);

System.out.println(list); 🡪 without toString()

**Internal working:** ArrayList<Integer> arr = new ArrayList<>();

arr 🡪 stored in stack

new ArrayList<>() 🡪 stored in heap

size is fixed internally but when array list fills by some amount then it will create a new arraylist that may be new size capacity and old elements copied to new list and old ones are deleted.

It has constant time complexity O(1).

ArrayList<ArrayList<String>> arr = new ArrayList<>();

[**Q9\_SwappingValuesInArray**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q9_SwappingValuesInArray.java)

[**Q10\_MaxValueFromArray**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q10_MaxValueFromArray.java)

[**Q11\_ReversingArrayValues**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q11_ReversingArrayValues.java)

**Linear Search Algorithm:**

**Q12\_FindWhether14ExistsInArray** – arr = [18,12,9,14,77,50]

Best case of linear search time complexity O(1)

Worst case O(N) 🡪 N is size of array

Best case going to be when searching for element is if element is at 0th place

How many checks will the loop make in best case i.e. element found at 0th index?

Linear time complexity is, when time is increasing linearly w.r.t size of the element.

[**Q13\_SearchInString**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q13_SearchInString.java)

[**Q14\_MinimumNumber**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q14_MinimumNumber.java)

[**Q15\_SearchIn2DArray**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q15_SearchIn2DArray.java)

[**Q16\_MaxValueIn2DArray**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q16_MaxValueIn2DArray.java)

[**Q17\_EvenDigits**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q17_EvenDigits.java) -> pending

[**Q18\_SingleNumber**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q18_SingleNumber.java) -> pending

**Binary Search Algorithm:**

Arr = [2,3,5,13,25,56,**64**,112,145,176]

0 1 2 3 4 5 6 7 8 9

Target element is 64 and assume this is an sorted array (ascending order)

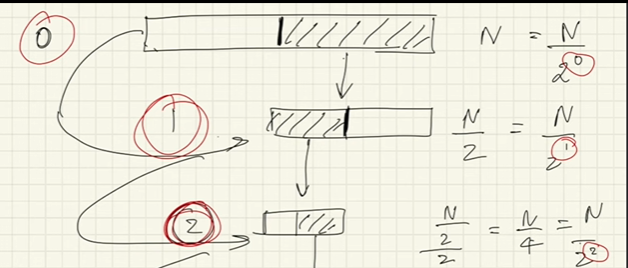
1. Take the middle of the element= 25 [(0+9)/2 🡪 4th index]
2. 64 is greater than 25 so target is in right half of the array only
3. Right side array = [56,64,112,145,176] middle (112)
4. And so on
5. If middle element = = target element then that will be answer

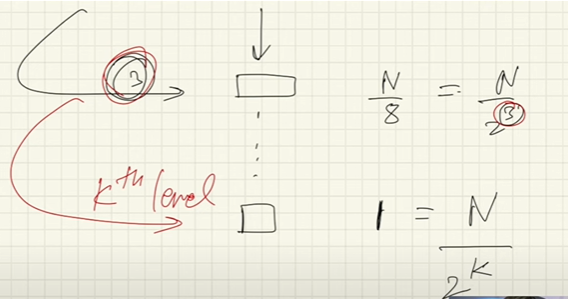
**Why Binary search:**

Best case scenario:

As the size increases time remains constant

Worst case scenario:





* N/2^k = 1 => N = 2^k
* Log(N) = k log(2)
* K = log(N)/log(2) -> ignore constant while doing space time complexity
* Total comparisons in the worst case = log(N)

Ex: search in a 100000 element array

Ans : log(100000) with base 2 -> 20 comparisons only

**#** There may be possibility that (start+end) thing can exceed the range of integer in java in this case mid = (start + (end – start)/2)

[**Q19\_BinarySearch**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q19_BinarySearch.java)

**Order Agnostic Binary Search:**

If we don’t know the array is sorted in which order.

Check first and last numbers to know order of array

**When to apply binary search:**

Square root of number

[**Q20\_CeilingOfaNumber**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q20_CeilingOfaNumber.java)

Ceiling number -> smallest number in array greater than or equal to target

[**Q21\_FloorOfaNumber**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q21_FloorOfaNumber.java)

Floor number ->. Greatest number in array smaller than or equal to target

[**Q22\_NextGreatestLetter**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q22_NextGreatestLetter.java)

[**Q23\_FirstAndLastPositionInSortedArray –**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q23_FirstAndLastPositionInSortedArray.java)

Sorted array so apply binary search

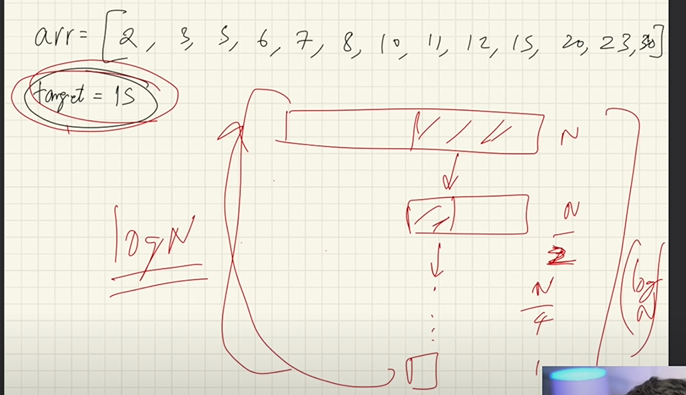
Run binary search 2 times (log(N) + log(N) = log(N)) no change in time complexity

Find first occurrence of target

[**Q24\_PositionOfElementInInfinteSortedArray**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q24_PositionOfElementInInfinteSortedArray.java)

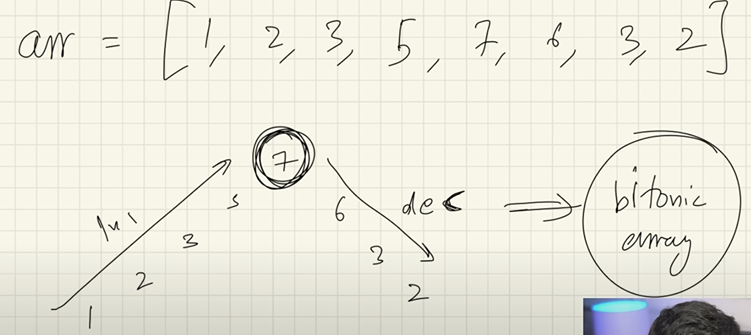
in case of infinite array avoid using length

here we will try bottom to up approach



[**Q25\_PeakIndexInMountainArray**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q25_PeakIndexInMountainArray.java)

Bitonic array



[**Q26\_SearchInMountainArray**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q26_SearchInMountainArray.java)

First find peak element , then find in left side else find in right side

**[Q27\_SearchInRotatedSortedArray](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q27_SearchInRotatedSortedArray.java)**

**Rotated Binary Search:**

Arr = {2,4,6,7,9,12,34};

After 1 rotation

Arr = {34,2,4,6,7,9,12}

After 2nd rotation

Arr = {12,34,2,4,6,7,9}

**Approach 1:** find the pivot in the array

Pivot: from where next numbers are ascending

If we are able to find pivot point and apply binary search in right side and left side

**Find pivot:** 1.apply binary search (mid > mid+1)

2. if mid < mid-1 is also answer

3. if start >= mid =>

Arr = {4,5,6,**3**,2,1,}

All element from mid(let 3) will be < start hence we can ignore all these elements since we are finding peak element

End = mid -1;

4. start < mid => arr = {3,4,**5**,6,2} -> sorted first half

Start = mid +1;

**Rotated Binary search for Duplicate value array:**

Arr = {2,2,2,2,9}

Arr = {2,9,2,2,2}

If elements at middle, start, end are equal then just skip the duplicates

**[Q28\_FindRotationCountInRotatedSortedArray](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q28_FindRotationCountInRotatedSortedArray.java)**

[**Q29\_SplicArrayLargestSum**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q29_SplicArrayLargestSum.java)  -- need to understand again

**Binary Search in 2D Array:**

[**Q30\_FindIn2DArray**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q30_FindIn2DArray.java) **(sorted row and column wise)**

(lower bound) Upper bound

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 20 | 30 | 40 |
| 15 | 25 | 35 | 45 |
| 28 | 29 | 37 | 49 |
| 33 | 34 | 38 | 50 |

Target =37;

We will search [row(0) , col(last)] 🡪 (40 -> 35 -> 37)

While(row < length && col >= 0)

Case 1 : if element is == target

Case 2 : if element < target

Target > 30 (we can ignore row 1)

Row++;

Case 3 : if element > target

Target < 40 (upper bound) -> this indicates 4th column will be cancel out

col--;

Time complexity : N + N = 2N -> O(N)

**[Q31\_FindInSorted2DArray](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q31_FindInSorted2DArray.java)**

RStart mid col

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| (middle row) 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

Rend

Target = 2

# take middle column and perform binary search ( can do for row also)

Case1: if( element == target)

Case2: if(element > target)

Ignore rows after it

Case3: if(element < target)

Ignore above rows

1. Check whether the mid col contain ans
2. Consider the four parts for remaining 2 rows
3. 1st row left
4. 1st row right
5. 2nd row left
6. 2nd row right

Time complexity: log(N) + log(M) = log(N)

**Sorting Algorithms:**

**Bubble sort/Sinking sort/Exchange sort**: every step comparing adjacent elements.

Arr = {3,1,5,4,2};

Here

1st 3 >1 🡪 swap {1,3,5,4,2}

3>5 🡪 as it is {1,3,5,4,2}

5>4 🡪 swap {1,3,4,5,2}

5>2 🡪 swap {1,3,4,2,5}

With the first pass through the array, the largest element come to the end.

2nd 1>3 🡪 as it is {1,3,4,2,5}

3>4 🡪 as it is {1,3,4,2,5}

4>2 🡪 swap {1,3,2,4,5}

4>5 🡪 as it is {1,3,2,4,5}

With 2nd pass 2nd largest element is at the 2nd last from the end

3rd 1>3 🡪 as it is {1,3,2,4,5}

3>2 🡪 swap {1,2,3,4,5}

3>4 🡪 as it is {1,2,3,4,5}

4>5 🡪 as it is {1,2,3,4,5} 🡪 **sorted**

With every pass we can ignore last parts that are sorted so in first pass we can ignore index 4, after second pass can ignore index 3 and so on

Space complexity = O(1) // no extra space required for copying the array

Time Complexity: **Best** case: O(N) => sorted (swap never happen)

**Worst** case: O(N2) => sorted in opposite

N – number of comparisons

1st pass N-1 comparisons

2nd pass N-2 comparisons

3rd pass N-3 comparisons

4th pass N-4 comparisons

Total comp = (N-1) + (N-2) +( N-3)+(N-4)

4N - (1+2+3+4)

4N – (N\*(N+1))/2

(7N – N2)/2

== O(N2)

# As the size of array is growing, the no. of comparisons also increases

[**Q32\_BubbleSortedExample**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q32_BubbleSortedExample.java)

**Stable and Unstable Sorting Algorithms:**

Arr = {10,20,20,30,10}

Arr = {10,10,20,20,30} 🡪 **stable sorted**

In original array black value of 10 was before red value of 10 and in sorted one this order is maintained

Arr = {10,10,20,20,30}; 🡪 **Unstable sorted**

**Selection sort:**

Select an element and put that on correct index.

Arr = {4,5,1,2,3};

{4,1,2,3,5}

{1,2,3,4,5}

**Time complexity:**

**Best case: O(N2)**

**Worst case: O(N2)**

Performs well in small arrays

**Q32\_SelectionSortExample**

**Q33\_SelectionSortExample**

**Insertion Sort:**

Sorting array partially

Arr = {**5,3,**4**,**1,2} -> first sort 5,3 (i=0,j=1)

Arr = {3,5,4,1,2} -> second 3,5,4 (i=1,j=2)

Arr = {3,4,5,1,2} -> third 3,4,5,1 (i=2,j=3)

Arr = {1,3,4,5,2} -> fourth (i=3,j=4)

Arr = {1,2,3,4,5} (i=4,j=5)

For every index : put that index element at the correct index of LHS